JDBC - STORED PROCEDURES

http://www.tutorialspoint.com/jdbc/jdbc-stored-procedure.htm Copyright © tutorialspoint.com

I have explained how to use **Stored Procedures** in JDBC while discussing JDBC - Statements. This tutorial is similar to that section but it would give you additional information about JDBC SQL escape syntax.

Just as a Connection object creates the Statement and PreparedStatement objects, it also creates the CallableStatement object which would be used to execute a call to a database stored procedure.

Creating CallableStatement Object:

Suppose, you need to execute the following Oracle stored procedure:

```
CREATE OR REPLACE PROCEDURE getEmpName
   (EMP_ID IN NUMBER, EMP_FIRST OUT VARCHAR) AS
BEGIN
   SELECT first INTO EMP_FIRST
   FROM Employees
   WHERE ID = EMP_ID;
END;
```

NOTE: Above stored procedure has been written for Oracle, but we are working with MySQL database so let us write same stored procedure for MySQL as follows to create it in EMP database:

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `EMP`.`getEmpName` $$
CREATE PROCEDURE `EMP`.`getEmpName`
   (IN EMP_ID INT, OUT EMP_FIRST VARCHAR(255))
BEGIN
   SELECT first INTO EMP_FIRST
   FROM Employees
   WHERE ID = EMP_ID;
END $$

DELIMITER ;
```

Three types of parameters exist: IN, OUT, and INOUT. The PreparedStatement object only uses the IN parameter. The CallableStatement object can use all three.

Here are the definitions of each:

Parameter	Description
IN	A parameter whose value is unknown when the SQL statement is created. You bind values to IN parameters with the setXXX() methods.
OUT	A parameter whose value is supplied by the SQL statement it returns. You retrieve values from theOUT parameters with the getXXX() methods.
INOUT	A parameter that provides both input and output values. You bind variables with the setXXX() methods and retrieve values with the getXXX() methods.

The following code snippet shows how to employ the **Connection.prepareCall()** method to instantiate a **CallableStatement** object based on the preceding stored procedure:

```
CallableStatement cstmt = null;
try {
   String SQL = "{call getEmpName (?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   . . .
}
```

The String variable SQL represents the stored procedure, with parameter placeholders.

Using CallableStatement objects is much like using PreparedStatement objects. You must bind values to all parameters before executing the statement, or you will receive an SQLException.

If you have IN parameters, just follow the same rules and techniques that apply to a PreparedStatement object; use the setXXX() method that corresponds to the Java data type you are binding.

When you use OUT and INOUT parameters you must employ an additional CallableStatement method, registerOutParameter(). The registerOutParameter() method binds the JDBC data type to the data type the stored procedure is expected to return.

Once you call your stored procedure, you retrieve the value from the OUT parameter with the appropriate getXXX() method. This method casts the retrieved value of SQL type to a Java data type.

Closing CallableStatement Obeject:

Just as you close other Statement object, for the same reason you should also close the CallableStatement object.

A simple call to the close() method will do the job. If you close the Connection object first it will close the CallableStatement object as well. However, you should always explicitly close the CallableStatement object to ensure proper cleanup.

```
CallableStatement cstmt = null;
try {
   String SQL = "{call getEmpName (?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   cstmt.close();
}
```

For a better understanding, I would suggest to study Callable - Example Code.

JDBC SQL escape syntax:

The escape syntax gives you the flexibility to use database specific features unavailable to you by using standard JDBC methods and properties.

The general SQL escape syntax format is as follows:

```
{keyword 'parameters'}
```

Here are following escape sequences which you would find very useful while doing JDBC programming:

d, t, ts Keywords:

They help identify date, time, and timestamp literals. As you know, no two DBMSs represent time and date the same way. This escape syntax tells the driver to render the date or time in the target database's format. For Example:

```
{d 'yyyy-mm-dd'}
```

Where yyyy = year, mm = month; dd = date. Using this syntax {d '2009-09-03'} is March 9, 2009.

Here is a simple example showing how to INSERT date in a table:

```
//Create a Statement object
stmt = conn.createStatement();
//Insert data ==> ID, First Name, Last Name, DOB
String sql="INSERT INTO STUDENTS VALUES" +
             "(100,'Zara','Ali', {d '2001-12-16'})";

stmt.executeUpdate(sql);
```

Similarly, you can use one of the following two syntaxes, either **t** or **ts**:

```
{t 'hh:mm:ss'}
```

Where hh = hour; mm = minute; ss = second. Using this syntax {t '13:30:29'} is 1:30:29 PM.

```
{ts 'yyyy-mm-dd hh:mm:ss'}
```

This is combined syntax of the above two syntax for 'd' and 't' to represent timestamp.

escape Keyword:

This keyword identifies the escape character used in LIKE clauses. Useful when using the SQL wildcard %, which matches zero or more characters. For example:

```
String sql = "SELECT symbol FROM MathSymbols
              WHERE symbol LIKE '\%' {escape '\'}";
stmt.execute(sql);
```

If you use the backslash character (\) as the escape character, you also have to use two backslash characters in your Java String literal, because the backslash is also a Java escape character.

fn Keyword:

This keyword represents scalar functions used in a DBMS. For example, you can use SQL function *length* to ge the length of a string:

```
{fn length('Hello World')}
```

This returns 11, the length of the character string 'Hello World'.

call Keyword:

This keywork is used to call stored procedures. For example, for a stored procedure requiring an IN parameter, use following syntax:

```
{call my_procedure(?)};
```

For a stored procedure requiring an IN parameter and returning an OUT parameter, use following syntax:

```
{? = call my_procedure(?)};
```

oj Keyword:

This keyword is used to signify outer joins. The syntax is as follows:

```
{oj outer-join}
```

Where outer-join = table {LEFT|RIGHT|FULL} OUTERJOIN {table | outer-join} on search-condition. For example:

```
String sql = "SELECT Employees
              FROM {oj ThisTable RIGHT
              OUTER JOIN ThatTable on id = '100'}";
stmt.execute(sql);
```